スクラム現場ガイド

MITCH LACEY[著]

安井力、近藤寛喜、原田騎郎[翻訳]

スクラムはシンプルだが
簡単ではない

スクラムの導入

＜スクラム＞
複雑で変化の激しい問題に対応するためのフレームワーク
可能な限り価値の高いプロダクトを生産的かつ創造的に届けるためのもの

- 軽量
- 理解が容易
- 習得は困難

以下のような変化を、チームが自発的に引き起こさなければならない

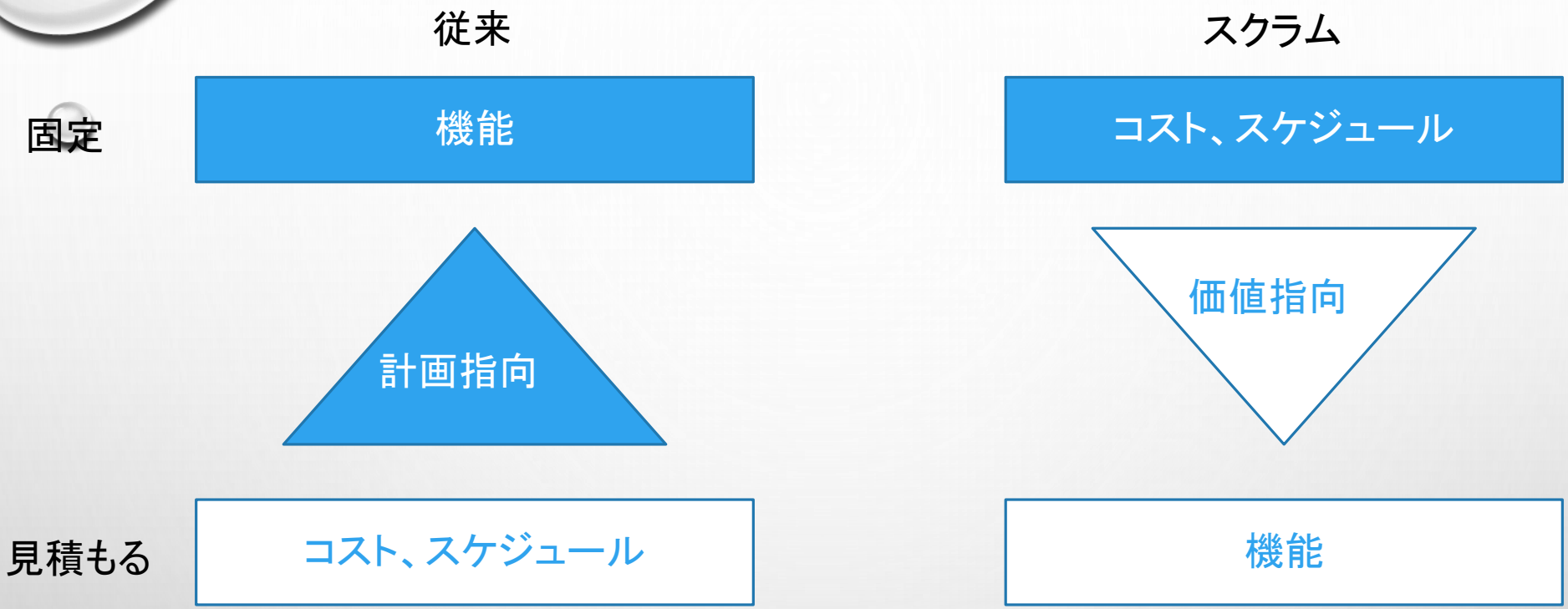
- マインドセットを変える。多くの場合、大幅な変化となる
- 変化を計画し、変化が起きたら適応する
- 新たに発見したり発生したりする問題に対応する
- アジャイルのエンジニアリングプラクティスを導入する

- 「問題を作りだしたのと同じ考え方で、問題を解決することはできない(アインシュタイン)」
  →マインドセットを変えられないことが最大の障害となる
- スクラムが通るのは最短の道、決められた道ではない
  →毎日、毎週、毎月のチェックポイントでチーム、マネジメント、顧客という様々なレベルの人々を集め、確認する
- 組織において当たり前になっていることを問いただすのがスクラム
- スクラムをプロジェクトをマネジメントするためのフレームワーク
  エンジニアリングを助けるのがエクストリームプログラミング(XP)

スクラムの5つの価値

＜集中＞
すべてを1点に集め、それだけに注意を向ける
集中している以上、プロジェクト一つしか、一度に進められない。「チームタスク」を設定して、
その間はメール、ｲﾝｽﾀﾝﾄﾒｯｾｰｼﾞ、TEL、ミーティングを禁じるという方法もある
チームが与えられたスプリントの間、完成に向けて全ての力と時間を集められるようにする
＜尊敬＞
与えられるのではなく、得るもの
チームの仲間を尊敬できるかどうかは、プロジェクトの成否を左右する
上手にやっているスクラムチームはお互いを信頼したっているので、オープンに障害物を認められる
真のスクラムチームには「自分たちとあいつら」という発想はない
＜コミットメント＞
誓約、約束、成果を届けるもの
コミットは気軽にせず、できるだけ情報を集めてからする
チームは組織に向かってコミットし、メンバー同士もコミットする
スプリントプランニングミーティングが終わったときにはチーム全体がスプリントで何を成し遂げるとコミットしたか、
チームメンバー全員が同じレベルで理解しなきゃいけない
＜勇気＞
困難に直面したとき、恐れを感じていたとしても勇気があれば立ち向かえる
恐れを減らし、チームメンバーに勇気を与えるために、チームや組織が最高の支援になる
ディスカッションで正直になることへの理解を示す
＜オープンさ＞
新しいアイデアを受け入れやすくなる
チームがオープンであることを示すいい機会がスプリントのふりかえり
新しいアイデアや、ものの見方、考え方を受け入れようとする気持ちを持つ
学習する組織へと成長し、効果的なチームになれる

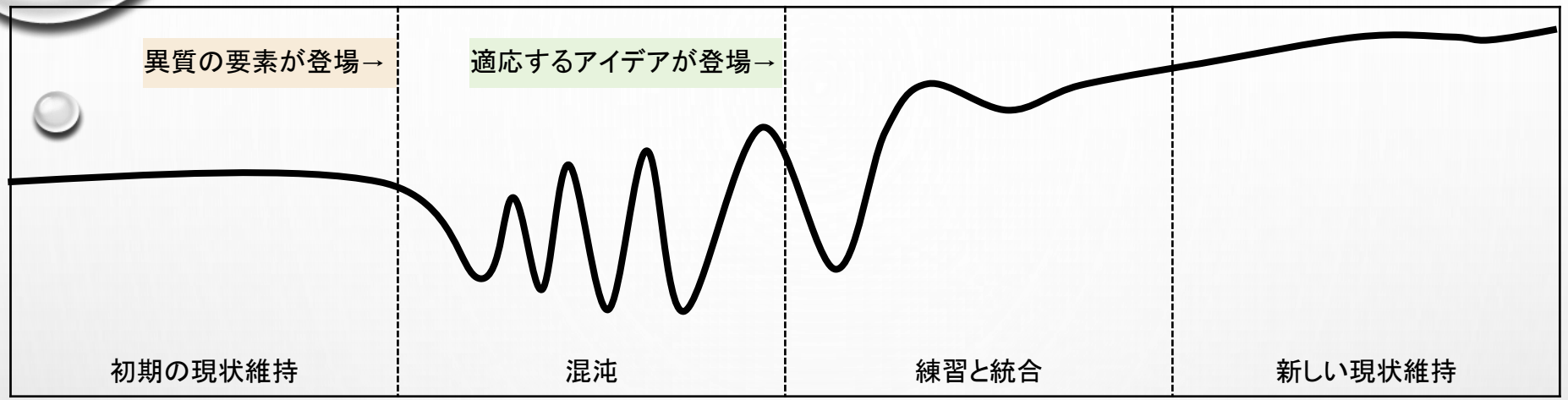
計画手法の対比



スクラムは一定の期日までに一定の機能を開発するという約束をしない
日付か機能のいずれかを可変にしておく

スクラムは変化のためのもの

サティアの変化のステージ



サティアの変化のステージ

初期の現状維持	誰でも居心地がよく、良く知っている世界であるため現状維持を続けようとすることが多い 変化の必要性を無視し、今までと同じことを上手に、懸命に、熱心にやろうとする デスマーチのようにチームは避けがたい失敗へ向かって突撃する 長時間残業し、品質を犠牲にしながら締め切りを守ろうと努力する やがて誰もが「必死で努力すればなんとかなると信じていて大丈夫か」と文句を言い始める 一部の人が今まで続けてきたやり方をやめたり、マネジメントが新しいやり方を導入したりする
異質な要素	異質な要素はチャレンジ いったん要素が現れると、もう無視することはできず、混乱が起きる 人々が恐怖を感じることが多く、変化が来るとわかっていても抵抗をする どう変化したらいいかわからない、期待がわからない、新しいやり方でも以前とおなじようにできるかわからない、 そういった理由から抵抗する
混沌	現状維持が異質な要素の登場で乱されたときに起きる 大幅な変更の場合には、知っていたものがすべて消えてしまうように感じることもある 新しいやり方が必要だと信じている人ですら、やり方が不確かで、不安だと感じる 混沌の状態では、ひとびとは神経質になり、不安を感じ、攻撃されているような気がすることもある 間違って動いてしまうことを恐れるため不安感が蓄積され、能力が大幅に揺れ動く 混沌ステージを通り抜けるまでは状況は一旦悪くなるがそのあと良くなる。適応するためのアイデアが見つかるまで混沌は続く
練習と統合	練習しなくては、異質な要素が異質でなくなり、馴染んだものになり、習慣化することはない 信頼、関係、個性がチームの中や、組織全体に生まれる 新しいやり方を学ぶにつれて、能力が向上し、変化以前よりよくなることもある 新しいやり方に自信をつけてきているが、不安な感じが残っている 新しい成功は一時的で、繊細なものなので大事に扱わなくてはいけない そうした成功を続けると行く先が見え始め多くの人にとって行く先は楽園のように感じられるが、 それでも混沌だと思う人も一部残る
新しい現状維持	適応するためのアイデアにチームメンバーが上達し、自信が持てるようになると以前を凌駕する能力を発揮する 元気が出てくるし、適応をやりとげたという達成感を持てる メンバーはさらなる改善を探り、物事をよりよくする権利があると感じ、学習する組織を作るため一致団結する 安全な環境があり、そこで練習を続けられる自由があることが大事

第1部
準備

仲間と共に旅立つ

チームコンサルタントでチームの生産性を最適化

ベロシティの測定

スクラムの役割

スプリントの長さを決める

完成を知る

専任スクラムマスターの利点

仲間と共に旅立つ

変化には時間がかかる

変化というのは大変
現状維持は悪かもしれないが、少なくとも既知の世界に居られる
未知の世界に飛び込むには勇気がいる

＜組織に変化を導入する8つのステップ※＞　※1995ハーバードビジネスレビュー、 ジョン・コッター

1	危機意識を生み出せ	新しいことに挑戦するよう人々を説得するにはなぜ取り組むべきなのか、その理由を伝える必要がある 気を付けないといけないのは、目標は達成可能であるべきで、緊迫感も現実に根差したものでなければならない
2	変革を進めるための連帯	スポンサー：組織内でプロジェクトの成功に対し、最終的な責任を負い、組織の境界や機能不全を壊していける人 よいスポンサーは変化を信じていて、チームを指導する力があり、中間管理者とビジョンを語り合い、なぜ変化が重要なのかを説明し、変化を効果的に起こすスケジュールを立てなければならない 中間マネージャーに新しいプロセスでの役割を説明し、実現のため彼らの助力を取り付ける必要もある 必要な場合は反抗しそうな人を見つけて懸念や恐怖を取り除くことまでする
3	ビジョンと戦略を作る	新しいことに取り組むには、その未来がどのように見え、どのように適合するのかを理解しなければならない
4	ビジョンを周知徹底する	本やホワイトペーパー、論文など優れた文献を紹介し、学びの機会を作る 何が試されてきて、どんな失敗があるのか、成功例や失敗例を理解できる 誰でも本から学べるというわけではないのでソーシャルな学習方法も検討する 自分自身の人脈を利用し、経験談を語ってもらう。 外部の人に話してもらうだけでも役に立つ。 人というのは知らない人の話を信じがちで、知っている人がおなじことをいうとかえって疑ったりする
5	従業員の自発を促す	投資額が小さいうちに失敗するほうが、後になって投資額が大きくなってから失敗するよりも望ましい プロジェクトの良い点にも悪い点にも素直になり、変化に対して前向きになる
6	短期的な成果の重要性	短期的な成功によりチームとステークホルダーの間の信頼関係が築かれる
7	成果を活かしてさらに変革を進める	始めてすぐに100%の成果が出るわけではないと理解してもらう。時間が必要 変化にはトレーニング、練習と時間が必要
8	新しい方法と企業文化	最初はつらいが、短期的な成功をつかみ、自分にとって向くやり方を見つけるにつれ、よりよくなってくる やがて、どうして以前のやり方をやっていたのだろうと思えるほど、新しいやり方が自然になる

成功のカギ

ボトムアップ活動は広がるのに時間がかかる
最も大事な成功のカギは「辛抱」と「情報」

辛抱強く	自分自身が納得していることを、他人が気づくように仕向けるのは、 実に難しい仕事のひとつ 解決策を人に押し付けるのも同じくらい困難 種を撒き、自然に成長させる 人はそれぞれ違う速度で、違うやり方で学ぶ 他の人のきっかけを探すため、人の話に耳を傾け、何を理解していないのか、 なにを怖がってるのか、なぜ変化をためらうのか、しっかり聴く 進歩がないように見えても、そこに焦りを感じないようにする 大きな価値を得るには時間がかかる
情報を提供する	前向きでない同僚に渡せる確かなデータを準備しよう 現在の状況がツライものであっても、ほとんどの人は動こうとしない もっといい方法があると本当に確信するまでは、穴から出ようとしない 新しい提案を分かりやすく伝え、人々が考えを変えるまで辛抱強く待つ

チームコンサルタントで
チームの生産性を最適化

チームコンサルタント

チームコンサルタントはチームとプロジェクトで足りないスキルを直接埋める
自分の専門性をチームに売るというサービスを提供
チームコンサルタントを実現するプロセスは2つに分かれている
　　チームコンサルタントプールを作る
　　チームを構成する

チームコンサルタントのプールを作る

手順1：移行計画を立て、伝える

普段から組織内で感じる問題を説明する
学習する組織になるべき理由をビジョンとして描き、伝える
現在の仕事や役割がなくなってしまうわけではないと理解してもらう
むしろ拡張される
今は、かつてないほど、個人とチームのコミットメントが顧客を満足させる力となっており
ひいてはビジネスの成功もコミットメントから導かれるのだという事実を強調する

手順2：一人ずつに選んでもらう

	向いている人	利点	欠点
チーム コンサルタント	• ソフトウェアアーキテクト • デザイナー・UI • テクニカルライター • 技術の専門家 • 開発マネージャー • ソフトウェアリーダ	• 1つの領域に特化 • 一匹狼 • 自分の専門分野で 人を助けられる喜び • その分野でのリーダー的地位 • 自分自身で管理	• プロジェクト完了時に 恩恵を受けられない • 新技術習得の機会が ほとんどない • よいサービスを提供できなければ ただのお荷物になる
コアチーム メンバー	• 多彩な能力のある プログラマーやテスター • スキルを向上したい個人 • 1プロジェクトに専念したい	• プロジェクト初期から完了まで参加 • 機能横断チームでの新スキル習得 • 他メンバーに新しいやり方を教え、 成長させる • プロフェッショナルおよび 技術者として成長	• チームメンバーとして 振舞える必要がある • プリマドンナには向かない

自分のチームを作る

手順1:プロジェクトに必要なスキルを把握する

能力	スキル	空き状況
チームプレイ	C#	
コミュニケーション	SQL	
顧客志向	Ajax	
衝突への対応力		
柔軟さ(学ぶ意欲)		

手順2:スキルや能力を一覧にして、人材をひとりずつ調べ、スキルの一覧に記入

	Aさん	Bさん	Cさん	Dさん	・・・
会社での立場	**開発**	**テスト**	**開発**	**ｱｰｷﾃｸﾁｬ**	
能力					
チームプレイ	★★			★	
コミュニケーション		★★		★★★	
顧客志向			★★		
衝突への対応力		★		★★	
柔軟さ(学ぶ意欲)			★★★★		
スキル					
C#	★		★★		
SQL			★	★★★	
Ajax		★★		★★	
UIデザイン	★★				
空き状況	★★	★★★★	★★★	★	

ベロシティの測定

チームのベロシティを見積もる

見積もるときのヒント

- プロダクトバックログを見積もる
- リファレンスストーリーを分割する
- 理想時間で概算する
- チームの稼働時間を見定める
- チームのベロシティを見積もる
- 幅を持たせたベロシティで話をする

ベロシティを見積もるアプローチの優先度

	過去のデータ	様子見	わからないが見積もる
既存のチームでよく知ったテクノロジー	**第1候補**	第2候補	第3候補
既存のチームでよく知らないテクノロジー	第2候補	**第1候補**	第3候補
新しいチームでよく知ったテクノロジー	第3候補	**第1候補**	第2候補
新しいチームでよく知らないテクノロジー	第3候補	**第1候補**	第2候補

ベロシティの係数表

		未知のテクノロジー			
			既知のテクノロジー		
カテゴリ	チーム/プロジェクトの構成	**低い係数**	低い係数	高い係数	高い係数
1	立ち上がったチーム	**0.85**	0.9	1.1	1.2
2	新しいチーム	**0.6**	0.8	1.4	1.5

スクラムの役割

スクラムの役割

役割	特徴
スクラムマスター	• 信頼を醸成する • 問題に気づくように促す • 影響力を通じてリーダーシップを発揮する • 人を相手にした仕事を好む • プレッシャーの中でも穏やかでいる
プロダクトオーナー	• 顧客の要望を聞いて、何が本当に求められているのか判別できる • ステークホルダーや顧客と機能について合意できる • プロダクトマネジメントに精通している • 基礎的な財務会計の経験がある • 開発中のアプリケーションが対象とする業界の経験がある
チームメンバー	• オープンなマインドの持ち主である • 改善したり人の改善を助けたりしたい • チーム指向である • 尊敬される • 謙虚である

プロジェクトマネージャーの仕事とスクラムの役割の対応

PMの仕事	スクラムマスター	プロダクトオーナー	チーム
顧客とコミュニケーションする		主	副
要求を管理する		主	副
予算を管理する		主	
チームのモチベーションを上げる	主		
プロジェクトのドキュメントを作る			主
問題や課題を解決する	主		主

スプリントの長さを決める

スプリントの長さを決める

スプリントとは刺激反応時間
　　プロジェクトの期間とスプリントの長さで
　　フィードバック/方向転換できるチャンスの回数が決まる

スプリントの長さを決めるために考慮すること
　　プロジェクトの期間
　　リスクを取れる量
　　チームの能力
　　顧客の忍耐

プロジェクトの期間が1年を超えるような場合には、プロジェクト自体を考え直す
複数年のプロジェクトは大きすぎるため短いリリースに分割する方法を考える
プロジェクトを分割する方法が見つけられないなら、会社には文化的な問題がある

完成を知る

完成の定義を作る

「完成の定義」を持っているのは
チームが最高の仕事でもってビジネスと顧客の期待に応えようとしている現れ

＜完成の定義を公開する3つの意味＞

1. チームメンバー同士の関係が深まる
2. ステークホルダーとコミュニケーションする役に立つ
3. チームが軌道を外れず、集中する助けになる

チームで完成の定義を作るためのやり方

＜準備するもの＞

チーム、　多数の付箋、ペン、2～4時間使える部屋、
オープンマインド、割り込みが入らないこと

＜ステップ＞

1. ブレインストーミング
   方向性を合わせるために質問を準備する
   「チームとして何をすれば、ソフトウェアを顧客やステークホルダーに提供できるか？」
2. カテゴリ分け
   どういうカテゴリで分類すればいいかを決める
   例.「開発、テスト、プロジェクトマネジメント、それ以外」
3. 整理と集約
   重複、類似、その他に整理
   類似　重なったものを読み上げて、「これはどういう意味？」と聞き、議論をまとめていく
   その他 読み上げ、意味を質問する。チームとして完成の定義に含めるかを決めていく
   答えないメンバーがいた場合は直接聞いて認識があっているかを確認。
   ※誰かを置いてきぼりにして次に進んではいけない
4. 完成の定義の作成と公開

専任スクラムマスターの利点

スクラムマスターの仕事

スクラムマスターがコストへ与える影響は大きい。会社のお金を節約できる
自分自身の本当の役割に気づくには、フルタイムで当たる必要がある
同時に、チームがポテンシャルを最大に発揮できるチャンスを与える

- 障害を取り除く、問題を解消する
- 争いや言い合いを収める/チームのママ役になる
- チームのデータ、パフォーマンスを報告する
- ファシリテーター役をする、必要になったら手を動かして手伝う
- 組織全体を教育する、組織的変化を主導する

第2部
現場の基本

エンジニアリングプラクティスのスクラムにおける重要性

チームのコアタイム

リリースプランニング

ストーリーやタスクを分割する

欠陥を抑制する

サステインドエンジニアリングとスクラム

スプリントレビュー

ふりかえり

概要(1/3)

エンジニアリングプラクティス※のスクラムにおける重要性

・スクラムはエンジニアリングプラクティスについて論じない
・高パフォーマンスを出すにはエンジニアリングプラクティスが必要不可欠

※テスト駆動開発、リファクタリング、継続的インテグレーションと頻繁なチェックイン、ペアプログラミング、統合と自動化された受け入れテスト

チームのコアタイム

・スクラムチームは出来る限り一緒に働く。同じ場所にいるのが望ましい
・コミュニケーションの効率、協力や連携、コードの共同所有も実現しやすい
・チームのコアタイムを決めるときに気を付けること
①チーム全員がコアタイムの意義を理解していること、
②コアタイムはプロジェクトの最初に設定し常に見直すこと、
③コアタイムが絶対ではないとチームに伝え状況に応じて対応できること

リリースプランニング

・スクラムプロジェクトでもリリース計画はあるが、アプローチが異なる
・見積もりを幅で表現して、自信の度合いを加えることで、アジャイルなリリース計画は従来手法の計画より情報が多くなる
・リリース計画は変更するために作ってあるのでステークホルダーやマネージャーもプロジェクトの状況が透明になり、向かう方向も、途中で何が起きるかも把握できる
・アジャイルなリリース計画を効果的に使うときに気を付けること
①包括的で頻繁なコミュニケーション
②スプリント毎のリリース計画更新
③優先順位最高のものから
④大きなものは再見積もり
⑤各スプリントの動作するソフトウェア提供

概要(2/3)

ストーリーやタスクを分割する

- スプリント期間内に終わらないのは作業を分解する能力が問題
- タスクの中に他のタスクを隠しながら、自覚していない
- ストーリーやタスクの大きさが適切か判断する一番の方法は質問
  - ①ストーリーは明確か？
  - ②ストーリーはどのくらい詳細か？
  - ③自分自身でできるくらいストーリーを理解しているか？
- 全員で見積もる。暗黙の仮定を見つける。自分自身でこなせるくらい理解する

欠陥を抑制する

- 欠陥対応コストは1:10:100ルールに従う(欠陥はその場で即時対応するのがベスト)
- 欠陥の優先度の基準を決めて対応していく。
- 欠陥マネジメントのプロセスを定義して欠陥が見えるようになれば、会社もチームも自分自身もウォーターフォール的な考え方から脱却できるはず

サステインドエンジニアリングとスクラム

- レガシーシステムをメンテナンスしながら新しい代替システムを作るのは大仕事
- 時間割り当てモデル
  1つのスクラムチームがプロダクトバックログを先に進めつつ、既存のプロダクトやシステムに関連する作業もこなす時間をかけてデータを集め、既存システムの対応時間をスプリントで使える時間から引いておく
- 専任チームモデル

概要(3/3)

スプリントレビュー

- 新しいチームはスプリントレビューで苦労する
- ストーリーを完成できなかった時や、思い通りに動かなかったものを見せるとなればスプリントレビューは無様で惨めな場となる
- スプリントレビューの準備をする※1時間以内に準備が終わることを目指す

スプリントゴール
コミットしたストーリー
完成したストーリー
完成できなかったストーリー
スプリント中の重要な判断(技術的なもの、市場起因のもの、要求に関するものなど様々)
プロジェクトのメトリクス(コードカバレッジなど)
完成したもののデモ
次のスプリントに向けた優先順位のレビュー

ふりかえり

- ふりかえりこそがチームの「検査と適応」のカギ
- 自分たちの仕事のやり方を見直す機会、チームの学びの時間
- どう改善するか、どう効率を上げるか、どう品質を上げるかを学び、常に改善し進化していくという意識を保つ

ふりかえり

ふりかえりの必要性

ふりかえりの原則
- ふりかえりはチームだけのもの。ふりかえりで出た話を外に漏らしてはいけない
- 小さな変化を積み重ねて自信をつける。
- 一度にたくさんやり過ぎるのはフラストレーションの元。
- 小さな変化をちゃんと計算しながら実現していけば自身が生まれる
- 変化は自分たちのもの。変化の責任はチーム全体にある。
- スクラムマスターだけの責任ではない。人々を巻き込んで変化を促進する

チームがプレッシャーを受けてスピードや効率や品質をあげたくなったときに
ふりかえりを止めてしまうが、ふりかえりこそがベロシティや品質を上げるための唯一の方法

ふりかえりをしないと重大な事態に陥る
- ほんとちょっとした不調(品質上の問題、通らないテスト、チームの士気低下)でも
- チームに蔓延し、伝染し、重篤化してしまう

ふりかえりがなければ、チームは自省できない。自省がなければチームは自分を失い
ある日突然、自分たちが変わり果ててしまっているのに気づく。身だしなみと同じこと

学びながら改善する心構えがないと、チームに古い慣習が戻ってきてしまう
そうして、どうにもうまくいかないと気づいてから、おかしくなったのはスクラムのせいだと文句をいったりすることになる

ふりかえりの実施

- グラウンドルールを壁に貼っておき、ミーティング開始時に読み上げる
  - 敬意を払う
  - 話をさえぎらない
  - ノートPCやスマホは片付ける
  - 長引く議論は「パーキングロット(駐車場)」に移す
  - 聞いた内容を言葉に出して、自分の理解を確認する
  - ここでの発言は外部に持ち出さない
- 最初の15分はデータ収集に充てる
  - スプリントバーンダウンチャート
  - 完成の定義
  - チームが描いたプロジェクトとスプリントそれぞれのビジョン
  - 前回のふりかえりのメモ
  - 「良かったこと」「改善したいこと」「不備」
- 問題に対する優先順位づけ
  - 各個人が持っているポイントを問題に割り当てる
  - 合計ポイントごとに問題を優先順位づけする
- 上から順にディスカッション
  - 対応するのかしないのか、対応しない場合は理由を残して共有する
  - 対応する項目には検討する人を1、2人割り当て計画をつくる
  - 問題が複雑なものは別のミーティングを開いてアクションを決める
- ふりかえりのクロージング
  - 1～5点で感想を教えてもらう

第3部
救急処置

生産的なデイリースタンドアップ

第4の質問

ペアプログラミング

新しいチームメンバー

文化の衝突

スプリント緊急手順

生産的なデイリースタンドアップ

開始も終了時間も時間通りに行う

・遅刻
　全員なんとしても時間通りに集まること
　遅れてきた人には失礼にならない範囲で他のメンバーが何分間ムダにしたと指摘する
　周囲からのプレッシャーが不十分だったらペナルティを科す
　ペナルティの設定の仕方には「自分の罰を自分で選ぶ」というのがある
　周囲のプレッシャーもペナルティも効果がなかったらよく遅刻する人と直接話し合う
・アジェンダ、リズム、レイアウト
　毎回必ず、スクラムマスターとしてミーティングを時間通りに終わらせる
　一人で話し続ける人がいる
　深堀したがる人がいる
　まとまりがない話をする人がいる
　デイリースタンドアップを習慣づけるためにはリズムが全て。下記をマントラのように毎日繰り返す
　「今日も集まってくれてありがとう。これから全員質問に答えます。
　①前回のミーティング以降何をしたか②今日これからなにをするつもりか③問題や障害が起きていないか。
　(④今回のスプリントゴールをチームで達成できる自信はどれくらいありますか？１～１０で教えてください)。
　さて誰から始める」
　参加者が一列になっていたら全員の顔が見えるように輪になる。輪になると順番が決まり進行がスピーディになる
　メンバーがお互いに向かって話すようになりスクラムマスターに向かって話すのを防げる
・割り込み
　トーキングオブジェクトを使って、オブジェクトを持っていたらしゃべっていい。持っていないなら話してはいけない。
・深堀
　目に見える「パーキングロット(駐車場)」にアイデアを留めて、ミーティングを前に進める。
　デイリースタンドアップ後の相談事項を残す

生産的なデイリースタンドアップ

隠れたインペディメントを暴く

デイリースタンドアップはチームが同期し、隠れた問題を明るみに出さなくてはいけない
進捗報告をするだけではなく、チームのコラボレーションを促進する場とする
スクラムマスターは前進に寄与しない発言に気を付ける

・まとまりがない

内容だけでなく、時間もまとまらないようにしないためにデイリースタンドアップでは質問を絞っている。
見つかった問題をその場で解決していてはダメ。デイリースタンドアップの準備をしてくるように指摘する。
準備できていなければ時間どおりに来ていてもいないのと同じ

・問題をごまかす

タスクがふくらんでしまうのはよくあることであり、それ自体は問題ではない
問題はなぜふくらんでしまったのか気づいていないこと

・曖昧

具体的にどんな作業をしているのかチームが理解する必要がある。曖昧な回答ではチームは何も学べない
透明性が失われれば、同じ作業を別の人が同時になってしまう事態も起こりえる

・深堀

目に見える「パーキングロット(駐車場)」にアイデアを留めて、ミーティングを前に進める。
デイリースタンドアップ後の相談事項を残す

第4の質問

「今回のスプリントゴールをチームで達成できる自信はどのくらいありますか？
1から10で答えてください」

この質問が効果的なのは、意見が強く、言いたいことを言える人物が会話やミーティングを支配してしまい、他のメンバーが「自分だけ違う、間違ったことをいいたくない、発言すると場を乱しそうといった感覚」で意見をいいづらいときに効果的
表面的な問題にとらわれずに深層に辿り着くには、対話へと導く質問が必要となる

第4の質問はプロジェクトへの自信に不一致があるときに、今まで気づかなかった問題を明らかにできる

個々人の作業から離れてプロジェクト全体のことを考える
感覚と直感を数字で表現する

第4の質問はスクラムマスターが以下のことに気を付けるとさらに効果的

＜非言語的コミュニケーション＞
チームメンバーのかすかなサインに気をくばる

＜衝突を避けようとする人＞
個性が文化の違いにより、衝突にどう反応するか大きく異なる。一人一人のことをよく知るようにする

＜継続的に学習し続ける練習＞
チームメンバーが自分自身を成長させ、能力を伸ばせるようなツールを提供すれば
現在だけでなく将来のプロジェクトまで続く大きな成果が得られる

ペアプログラミング

通常のペアプログラミング

一人がキーボードを持ち(戦術担当)、もう一人がそれを見ている(戦略担当)

戦術担当はキーボードを打ちながらタスクを進めているので外的な集中を維持しやすい

戦略担当は手を動かしていないために、内的な集中に問題が起きやすく、隙ができてしまうと生産性を低下させてしまう

プロミスキャスペアリング

上級スキルで経験を積んだ、能力の高いチームで実施

チームによるタスク所有で、タスクの担当者を決めない。いろいろなペアが入れ替わり立ち代わり作業する

スイッチチームのペア枠と入れ替わりの様子

	ペア枠1 10:30–12:00		ペア枠2 13:00–14:30		ペア枠3 14:30–16:00	
	メイン	サブ	メイン	サブ	メイン	サブ
PC1	A	B	B	C	C	D
PC2	C	D	D	E	E	F
PC3	E	F	F	A	A	B

マイクロペアリング

①一人がテストを書き、キーボードを渡す

テストが失敗したら、受け取った人が実装し、テストを通す

テストが通ったら、受け取った人はテストを追加するか、リファクタリングする

②キーボードを元に返して、テストを追加するか、今書かれたコードをリファクタリングする

③このパターンをタスクが完了するまで続ける

新しいチームメンバー

タックマンによるグループの進化のステージ

＜形成期＞

チームはどうやって一緒に働くかを学んでいく。お互いにどんな人か知る。
政治と個人主義が重要で自分自身の力量を見せて立場を作ろうとする。

＜混乱期＞

チームは結束を見せ始める。個人間の問題が解消し始める
衝突はなくならないものの、チーム全体が協力して問題にエネルギーを注ぎ
一緒に問題解決にあたるようになってくる

＜統一期＞

チームは一緒に働くのが上手になる。
チームのプロセスが第2の本能になり、自分たちの進め方を信じられるようになる。
メンバーはお互い親しくなり、批判も破壊的・政治的なものから建設的なものになる。
自分たちで設定したルールを守り、機能するチームへ近づいていく

＜機能期＞

チームは高度な信頼に到達する。チームの働き方は自然でなめらかになり、フロー状態やゾーンに入れる
関係が強まり、批判が自然になる。
問題にはチームで取り組み、個人が弱みを見せたり助けを求めるのに抵抗がない。
チームは日々進歩する

チームを形成しなおすのは大変。新たなメンバーをチームの一員とするのに大事なこと

①人選

候補の個人的な価値観に注意する
変化に前向きか？柔軟か？謙虚か？チームプレイヤーか？学ぶ意欲があるか？

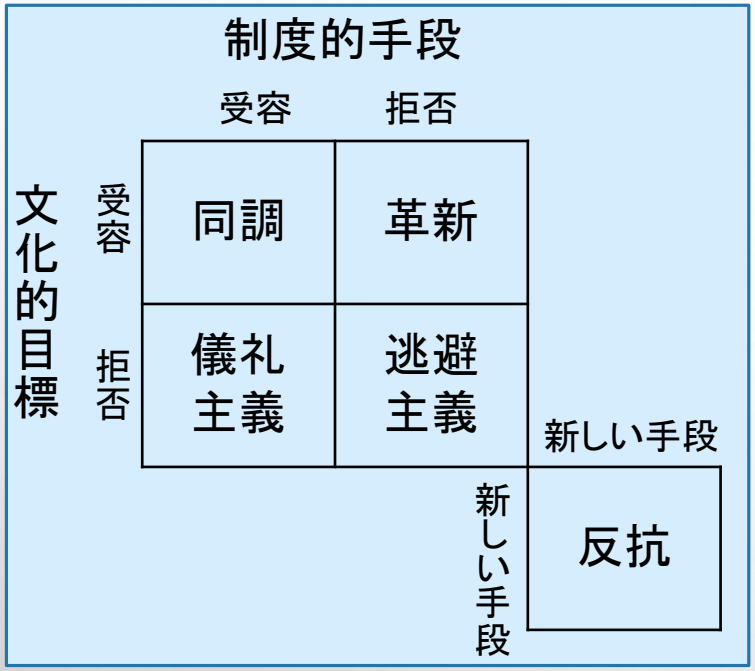
②①の回答がすべてYesならスキルを見る

スキルは価値観より伝えるのが容易なのでスキルより価値観を優先する

新メンバーを迎えることになったら、一時的にチームが後退するのもやむを得ないと受け入れ注意深く人選する

文化の衝突

ロバート・K・マートンの逸脱行動のトポロジ



＜同調＞
チームメンバーは制度的手段によって会社のゴールを達成する
- 高品質、士気向上など
- 従業員はプロセスに納得している
- 人々はやる気がある

＜儀礼主義＞
チームメンバーは制度的手段を受け入れるが会社のゴールを達成できない
- 低品質、士気低下など
- 言われた通りやればいい
- やる気を失っている

＜逃避主義＞
チームメンバーは手段を受け入れず、会社のゴールも達成できない
- 低品質、士気低下など
- 会社にはいるが仕事をしない
- わざとやる気なく振舞う

＜革新＞
チームメンバーは新しい手段をつくり、会社のゴールを達成する
- 消極的な場合も積極的な場合もある
- 積極的であれば高品質、士気向上など
- 反抗と異なりゴールは変わらず手段だけが変わる
- 組織において摩擦が起きる

＜反抗＞
チームメンバーは新しい手段をつくり、会社のゴールを元に新たなゴールを設定
- 手段はまったく変わってしまうかもしれない
- 会社のゴールは達成される。チームはそれを上回るゴールを設定
- チームの文化はまったく異なるかもしれない
- 組織において摩擦が起きる

文化の衝突

文化の衝突に対処するには、チームに合わないメンバーを最初から入れないのが一番

理想的なチームにはチームの運命はチーム自信が制御すべきであり、チームの構成も自分たちで決めるのが一番よい
理想的とは言えないチーム構成になったときには障害に直面しながらコースを外れないようにできることをやるしかない

逸脱の可能性がある場合には以下のガイドラインを参考にしてチームの高効率を守る

- 逸脱者をチームから外す。マネジメントには人が交換可能な部品ではないと理解してもらう。
  チームメンバーは気を付けて選ぶべきで、アサイン状況やスキルだけで決めてはならない

- 新メンバーにチームの文化を教え、適応する時間と空間を与える。そうして反抗や逃避を防ぐ

- 新メンバーがチームの基準から逸脱し始めたら、基本に戻りスクラムに馴染ませる
  これに失敗したら、新メンバーがなぜチームの文化から外れているのか理由を探る

- 新メンバーの逸脱から新たな視点が得られ、そのおかげでチームに良い変化が起きることもある。チームには自分たちにも適応する必要があると理解してもらう。
  チームの構成が変わったら、これまでのやり方を修正して新メンバーを受け入れなければならないこともある

- 逸脱行動が続くなら、人事やマネジメントやチーム自体を巻き込み、そのメンバーを他のより適切なチームへ異動するように働きかける

スプリント緊急手順

スプリントゴールを達成するためのチームの能力に影響がある想定外の事象が発生した場合、チームには4つの選択肢がある

1．障害を取り除く
2．助けを得る
3．スコープを減らす
4．スプリントをキャンセルする、中止する

どの選択肢も痛みを伴うが、いくつかのルールに従えば痛みを最小限にできる

＜コミュニケーション＞

スクラムの価値を活かせるのがコミュニケーション
困難を伴う仕事を進めるには勇気が必要であり、コミュニケーションも頻繁に必要

＜落ち着く＞

顧客やステークホルダーの反応をマネジメントしなければならない
正しい道を行き、スクラムの価値を説明するチャンスにする
スクラムであれば、選択肢と柔軟性が手に入る
短いサイクルのため、正しい道に素早く戻れる

＜集中する＞

後ろではなく、前を向く
スプリントを片付けて、やるべきことを淡々とやるよう集中する
あなたの反応が、他の人の反応を決める

第4部
上級サバイバルテクニック

持続可能なペース

動作するソフトウェアを届ける

価値の測定と最適化

プロジェクトのコストを事前に備える

スクラムにおけるドキュメント

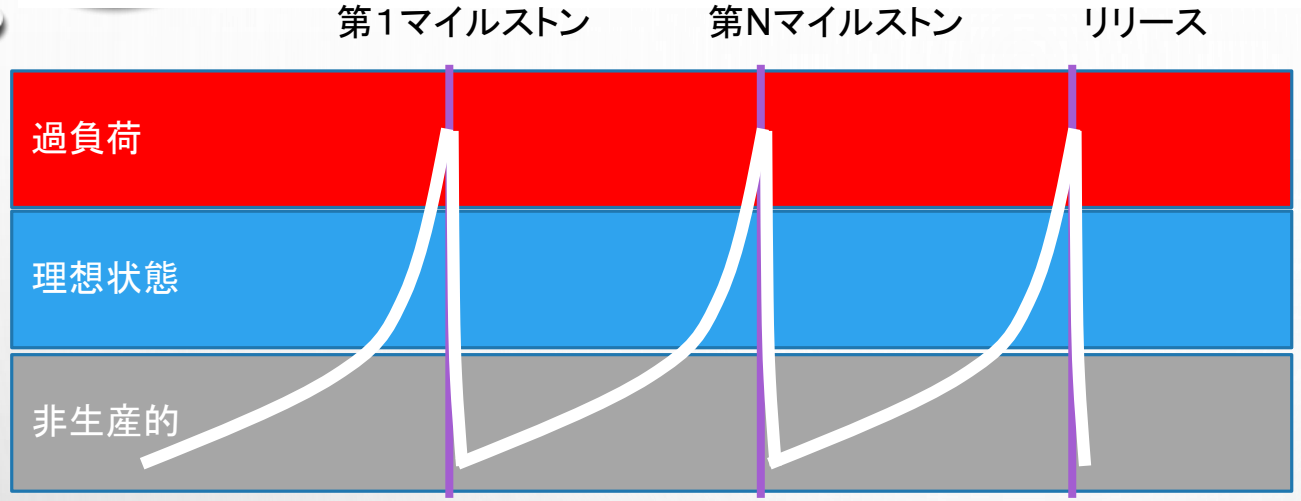
アウトソースとオフショア

巨大なバックログの見積もりと優先順位づけ

契約の記述

持続可能なペース

ソフトウェア開発サイクルのマイルストーンのおける山と谷



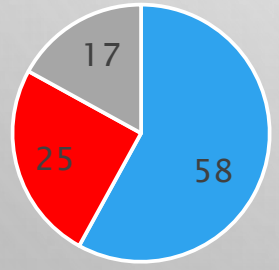
マイルストンを達成したり、スプリントレビューが完了するとチームが非生産的な状態になる

それからしばらく理想の効率で働いた後、次のゴールを達成するために過負荷になる

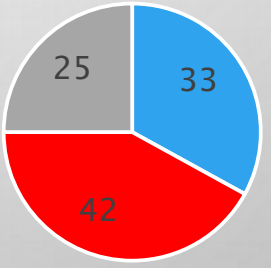
目標が達成されるとチームは一息ついて次の加速に備えるため再び非生産的な状態に戻る。

最初の何サイクルかは、非生産的な状態、過負荷な状態よりも理想状態で働いている時間が2倍はあるから
チームが痛みを感じることはあまりない。
マイルストンが進んでくると分布がずれはじめ過負荷で働く時間が長くなり、回復の時間も長くなり
理想状態で働ける時間は短くなり、悪循環にとらわれる。
サイクルが1ヶ月以上の場合はよく起こる

第1
マイルストン



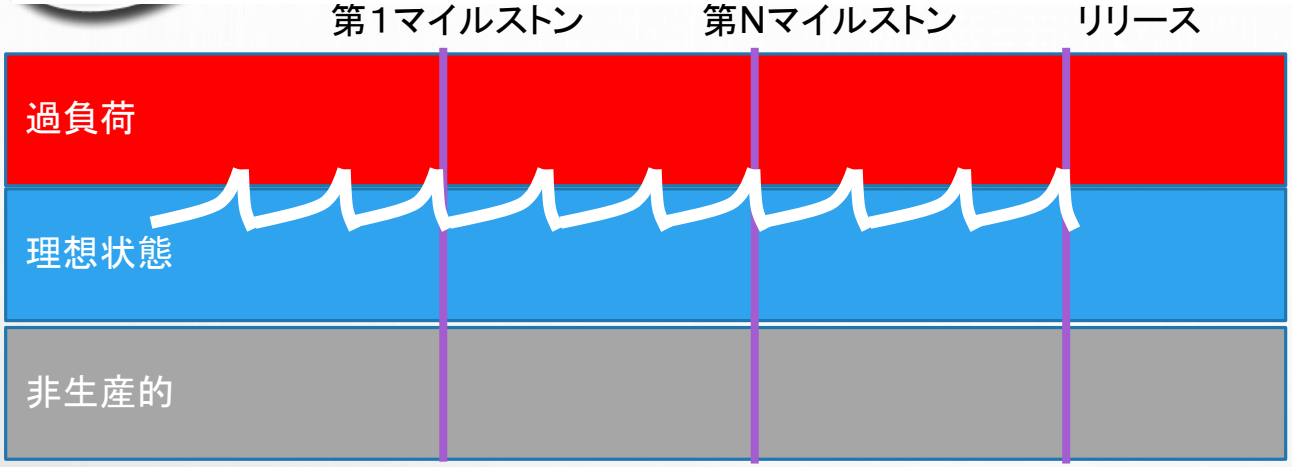
第N
マイルストン



https://coredou.com/

理想的なペースで働く

提供が頻繁にある山と谷



従業員を残業させ、燃え尽きさせてしまうひどいペースから理想的で生産的な継続可能なペースに移行するには
練習、トレーニング、規律、時間が必要

・イテレーションを短くする
　イテレーションを短くするとエネルギーを急に使うのも、それから回復するのも短くて済む
　スプリントレビューやふりかえりでこまめにチェックし、調整される
　心理学的には、チームに最初からゴールが見えていることでゴールにむけて集中しやすくなる

・バーンダウンチャートを監視する
　チームが過負荷の状態になると、停滞していた活動が急激に降下(過負荷状態)する
　メンバー一人が仕事を抱え込んでいる/タスクのサイズが不適切/スプリント中に集中できていないなど
　急降下を評価する時間を取ってひとつずつ修正していく

・一緒に働く時間を長くする
　コア時間をチームで維持するのは持続可能なペースを保つためにも欠かせない
　距離が増えると注意が削がれコミュニケーションが減る
　適切なタイミングで情報共有がされないとスプリント終わりの急降下につながる

動作するソフトウェアを届ける

「完全な、リリース可能なコンポーネント」ではなく、
動作するエンドツーエンドのシナリオにこそ価値がある

毎スプリントの動作するソフトウェアを根付かせるには基礎となるコアストーリーかユーザーの数のどちらかに着目する

＜コアストーリー＞

コアストーリーがシステム全体を通り抜けるようにして、他の可変要素は固定してしまう
プロジェクトが進んだら、モックやファイルに代わるコンポーネントを色々追加する
動作するソフトウェアを顧客に届け、顧客のニーズにあっているかを検証し、それを受けて計画を練り直す

＜ユーザーの数＞

システムを利用できるユーザーの数を制限したうえで、エンドツーエンドの機能を実現する
システムの成長とともに対応するユーザー数を増やしていく

考え方を変化させる

従来は複数のコンポーネントを並行開発し、プロジェクトの終盤ですべて結合してきた
このような開発の問題点は個々のコンポーネントに価値がないこと、コンポーネントは結合して始めて価値が生まれる
紙幣は破くのではなく、硬貨に両替するように順次開発という枠組みから離れる方法を考え抜く

手戻り

動作するソフトウェアを届けるためにファイルやモックオブジェクトを使うといずれリファクタリングが必要になり
バックエンドの機能を充実させるにつれて、一部のコードは破棄することが出てくる
予測できる手戻りもあれば、予測できない手戻りもある
完璧なシステムを1年後にリリースして想定通り動かなかったり、顧客の期待どおりでなかったり、
時代遅れだったりすることに比べれば、手戻りがあるほうがはるかにいいと思える

価値の測定と最適化

どんなにプレッシャーをかけられても、それ以上ベロシティを向上できなくなるときがくる
顧客とステークホルダーは裏方仕事をまったく見ないし、出したお金に対してできるだけ多くの機能が欲しい
機能実現の直接的な作業以外について、実際に何をしているのか洞察をし、顧客やステークホルダーに対して透明性が必要
作業を以下の5カテゴリに分類し、データを構造化する

＜機能実現＞

システムやソフトウェアパッケージのユーザーや購入者に対してビジネス価値を届けるために仕事

＜税＞

会社のための仕事や必須の要求事項であり、チームやグループに重荷となる賦課、義務、任務、請求など

例）セキュリティレビュー、法規コンプライアンスレビュー、スクラムミーティング、全体会議や部門会議

ソフトウェアのリリースにまつわるものもあれば、チームの時間を奪うものもある

＜スパイク＞

スパイクは短時間、タイムボックスで行う活動で、大規模だったり曖昧だったりするタスクやストーリーを
完成するのにどんな作業が必要かを見つけるためのもの

スパイクはタイムボックスで一定時間をあらかじめ定めておき、その中で探索し、見積もりが難しい項目を今よりしっかりと定義する
スパイクで見つけたタスクやストーリーは、現在のスプリントバックログに乗せず プロダクトオーナーが優先順位を付ける

チームは現在のゴールとスプリントバックログにコミットしている。
スパイクもその中に含まれるが、スパイクで見つかるタスクにはコミットしていない
スパイクの結果をスプリントに入れるため、あらかじめ時間を確保しておいたらスパイクのメリットを失う
新しいタスクは必ずしも優先順位が高いとは限らない

＜前提条件＞

前提条件はストーリーに関連するタスクではないが、スプリント中に終わらせるべき作業のこと
チームが見つけ、プロダクトオーナーやスクラムマスターと交渉する
これらの作業が完了しなければチームはスプリント完了と言えない
例)ビルド環境をつくる、テストフレームワークを準備する、モラル構築や職場環境の整備

＜欠陥/バグ＞

プロジェクトのコストを事前に考える

プロジェクト開始前にコストを判断するのはスクラムでも難しいが答えの伝え方が変化する

どのくらいコストが掛かるのか？この機能はいつ完成するのか？得られるものは何なのか？ どのくらい時間が必要か？

⇓

コストの議論はスプリントあたりのコストへ変化
機能仕様書は見積もって優先順位を付けたプロダクトバックログに置き換わる
時期と金額の質問は予算とベロシティの質問となる

コスト計算そのものはそう複雑ではなく、難しいのは信頼の構築

これまでの経験で「ソフトウェアを提供されない」、「期日を破られる」、「品質が期待以下」という目に遭っている
コストの質問は具体的に見えるが、実際には「依頼したものが本当に提供してもらえるのか、どうしたらわかるんだ？」という質問
質問に答えるときは、本当に示しているのは保証だと肝に銘じ、時間を使ってソフトウェアをどう開発し、提供するのか説明する
回答はすべて、実現できると考えられる幅で示し、確定の数字は使わない
チームの助けがない場合や実データで見積もっていないベロシティを使うときは仮定を伝え、後で回答が変わると理解してもらう
そのときには、実データがわかり次第速やかに共有すると約束する
変化は起きるものであり、いま現在の回答は推測に過ぎず、確定でも約束でもないと理解してもらう

アジャイルな計画づくりはわかりやすい。
顧客やステーク、マネジメントに事前に把握した情報を伝え、その後変化した内容も伝え、
動作するソフトウェアをスプリントごとに提供し、調整ができるようにする
これがスクラムであり、アジャイル。これこそが信頼を培うプロセス

コスト計算のステップ

1. スクラムのプロジェクトでは機能仕様書を使わず、ユーザーストーリーとプロダクトバックログを使う
   抽象度が高いままで機能を記述し、何を構築するか把握するのに十分な情報を提供しながら
   実装のタイミングまでは詳細に踏み込まないが以下の①、②が必要
   ①事前の情報を最小限にする方針、②チームがプロダクトバックログをスプリントごとにグルーミングするという約束

   ＜機能仕様書＞
   システムの機能の詳細を記述するだけでなく、
   システムのアーキテクチャや構成要素の詳細も記述されている
   もっとも情報が少ないプロジェクトの最初期で記述する
   計画も書いてあるが作る側の能力ごと(設計、開発、テスト)に作るよう想定しており、
   システムが提供する機能ごと(機能X、機能Y、機能Z)に作るとは想定していない

2. ユーザーストーリーを作る
   ユーザーストーリーは抽象。抽象は詳細よりも長生きする
   機能仕様書と違って、詳細を省いて将来プロダクトオーナーとチームの会話によって担保する
   詳細はストーリーの実装スプリント直前になってから会話を通じて示し、確認する

3. ストーリーを見積もる
4. ストーリーに優先順位を付ける
5. ベロシティを決める
6. コストを計算する
7. リリース計画をつくる

スクラムにおけるドキュメント

アジャイルソフトウェア宣言から『アジャイル開発にはドキュメントがない』
という誤解があるがこれまでと変わらず価値を置いている

アジャイルソフトウェア開発宣言

私たちは、ソフトウェア開発の実践
あるいは実践を手助けをする活動を通じて、
よりよい開発方法を見つけだそうとしている。
この活動を通して、私たちは以下の価値に至った。

プロセスやツールよりも**個人と対話**を、
包括的なドキュメントよりも**動くソフトウェア**を、
契約交渉よりも**顧客との協調**を、
計画に従うことよりも**変化への対応**を、

価値とする。すなわち、左記のことがらに価値があることを
認めながらも、私たちは右記のことがらにより価値をおく。

ドキュメントを書くタイミングを「開始時や終盤に重点的に書く」のではなく、「進みながら書く」ことで
知識のレベルとドキュメントを一致させるようにしただけ
適切なタイミングで、正確に、責任を持ってドキュメントを書く

ドキュメント作業はコードや自動化と同等の扱いで完成の定義に加える
変更が起きたら変わるのはコードだけじゃなく、ドキュメントも変わる

ドキュメント作成のカギ

判断

このプロジェクトでは何をドキュメントにするか、いつドキュメントをつくれば最適かを決める

コミット

ドキュメント作成の計画を決めたら、その計画を守る
完成の定義に追加し、あなた自身の信頼性を維持する
ドキュメント作業はいくら小さくしても楽しくないが、終盤のリリース時期に大きなリスク削減になると納得する

コミュニケーション

プロジェクトが包括的な事前ドキュメントをなくす初めての試みなら参加者を支援する
プロジェクト開始時には頻繁に状況の変化を伝え、ホワイトボードの写真など様々なドキュメントを作り次第送る
動作するソフトウェアと、モノとしての成果物を提供し続ければ心配性なお偉いさんの恐怖感も和らぐ

自動化に投資する

ドキュメント作業や自動化に少しでもいいので時間を投資する
自動スクリプトをつくったり、機能そのもの※を自動化する　　※例：インストールを手作業ではなく、自動化
ドキュメントにせよ、機能にせよ、自動化すれば投資に見合う効果が得られる

アウトソースとオフショア

作業が山積みになったり、コストが高まってくると、作業の一部を外部のチームに依頼したくなる
海外オフショアへのアウトソース開発は悪評にも関わらず普及し、理想的とは言えない状況で多くのひとがベストを尽くしている
マネジメントはいくつかの選択肢を検討したうえでアウトソースが最善だと判断するが
考慮が漏れがちなのが、チーム、コード、完成品、最終的な収支

本当のコストを考える
　時給だけでなく、隠れたコストを考慮に加える

＜引き継ぎのコスト＞
　今のチームとの共同作業の時間、ペア作業の時間
　プロジェクトの大きさ次第で知識を引き継ぐのは数週間～１年かかるかもしれない
　引き継ぎのコストはプロジェクト全体コストの5～57%を占めるという調査がある
＜オーバーヘッドの増大＞
　コミュニケーションの課題が発生し、生産的な時間は減る
　追加のミーティング、メール、電話。文化とコトバの壁
＜長期的な離職率＞
　海外チームは離職率が高め。仕事ができるようになると自分には価値があるとわかり別の仕事に移ってしまう
　引き継ぎ時間が長いと、もともとのメンバーがプロジェクトに意欲を持ち続けるのは困難になる
＜文化的課題と仕事管理＞
　ミーティング/電話/テレビ会議などのひとつひとつの調整、メールの文面の詳細化
　IBMのヘールト・ホフステッドが発見した「多文化の5つの主要な要素」に代表される異なる文化の理解
＜開発プラクティス＞
　新チームにもアジャイルなエンジニアリングやベストプラクティスを用いた同じレベルの仕事のやり方を期待する が
　そうしたチームや安くは雇えない
　プラクティスを実施していないチームを教育するのも時間とお金がかかる

アウトソース、オフショアの成功のカギ

正しいアウトソース先チームを選ぶ

- アジャイルプロジェクトをアウトソースするならアジャイルチームを雇う
- アウトソースするアジャイルの考え方が自分たちのチームと一致しているのかを確認する
- 時差が少ない南北で探す

苦痛を最小化するように仕事を振り分ける

- 一般的なやり方として「ワークパッケージ※」をチームの成果物とする　※機能、あるいは機能のセット
- エンジニアリングプラクティスと規約や規律を決めて、複数チームで同じコードベースをさわる
- アウトソース先を独立したチームとし、完全機能や機能群を任せ、エンドツーエンドで完成してもらう
- テスト、UI、システムの一部レイヤーなど特性の作業領域のみ割り振るのはフィードバックや結合までのループが長くなるのであまり良くない

スクラムのフレームワークから離れない

- デイリースタンドアップを続ける
- リアルタイムでテレビ会議でスプリントレビューを実施
- ふりかえり

ワンチーム文化を育てる

- ペア作業で共通理解を養う
- 継続的インテグレーション
- 常につながれるようなテクノロジーを利用する

出張に備える

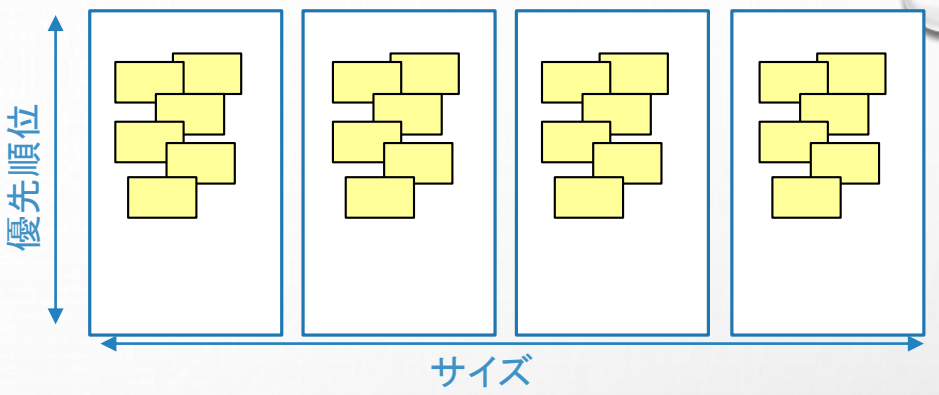
- 出張のための予算を組む
- 信頼関係ができるまで一緒に仕事をする。信頼にまつわる問題が表面化したら何度も実行する
- 時間のムダ、信頼のないチームが浪費する時間のコストをなくす

プロジェクト/チームの調整役を立てる

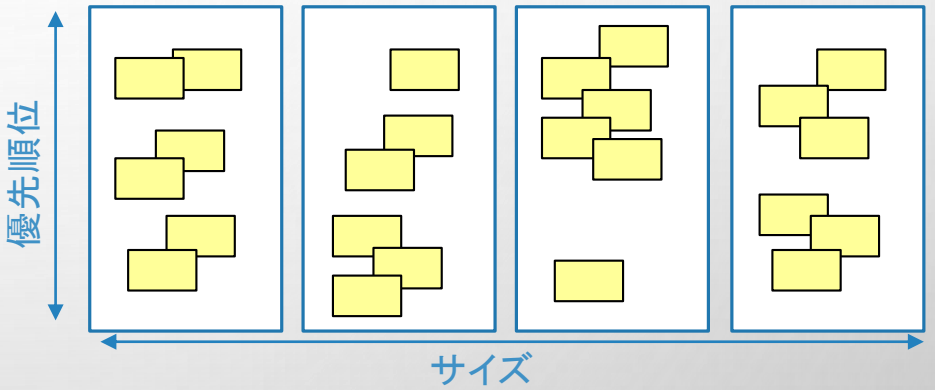
巨大なバックログの見積もりと優先順位づけ

広大な壁を準備してステークホルダーとチーム全員を集め丸一日でストーリーに立ち向かう

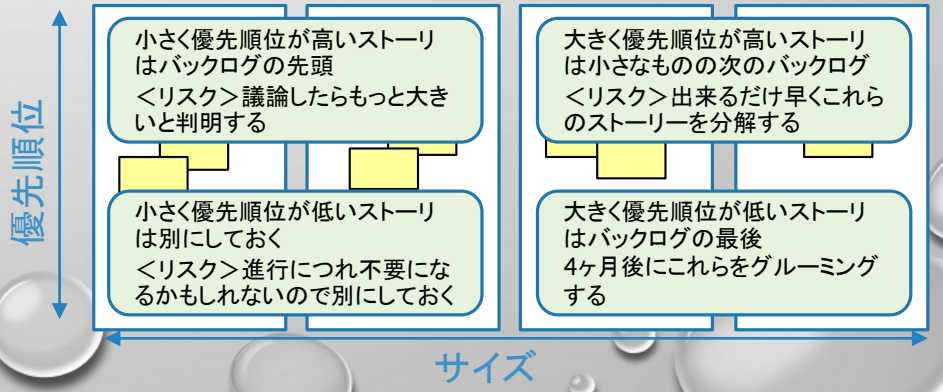
チームに指示して一番左に最小のストーリーを置き、
一番右には最大のストーリーを並べる



ステークホルダーに集まってもらい理由を伝え
達成したい目標を説明
ストーリーを上下にテープの区切りは超えずに移動してもらう



壁全体を見直す



契約の記述

通常の契約モデルでは、顧客は何を、いくらで、いつ手に入れるのか記述した契約書が必要
スクラムではスコープ、コスト、納期のうち固定するのは一つで他は変数にしようとする

顧客は最大限の機能と品質を、最小限のコストと最短の納期で欲しがる
顧客はプロジェクト開始前から100%の保証を求める

約束どおりにソフトウェアを届けられなくなる理由は①変更と、②タイミングの2つ

これまでの契約指示と変更指示

1. 会社、顧客は提案依頼書(RFP)を公開/送付
   RFPには顧客が必要な機能/作業が全て記述されており、通常優先順位はなく、すべて必要

2. 請負業者には短い質問の時間が与えられ、詳細な質問をしたり、想定条件を具体化したり予算を予測する
   RFPを作成するときは全体のプロジェクトを顧客の予算に合わせる

3. 最終的な提案は、ビジネスで勝つために顧客が欲しがるものを全部詰め込む
   顧客予算範囲の下限にできるだけ近づけ、競合他社より低くなるようにする

4. 提案には、安く出しすぎた提案をビジネスにするため変更が必要とわかったときの対処を決める**変更指示手順**を含める
   プロジェクト開始前の短い期間で顧客の要望を全て見出すのは不可能。物事は変わる

変更指示手順

変更要求は甚大なコスト増を生み、ベンダーにとって負担を強いることになる
ビジネスを続けるためベンダーは大規模で詳細な変更指示手順をつくり、徹底する

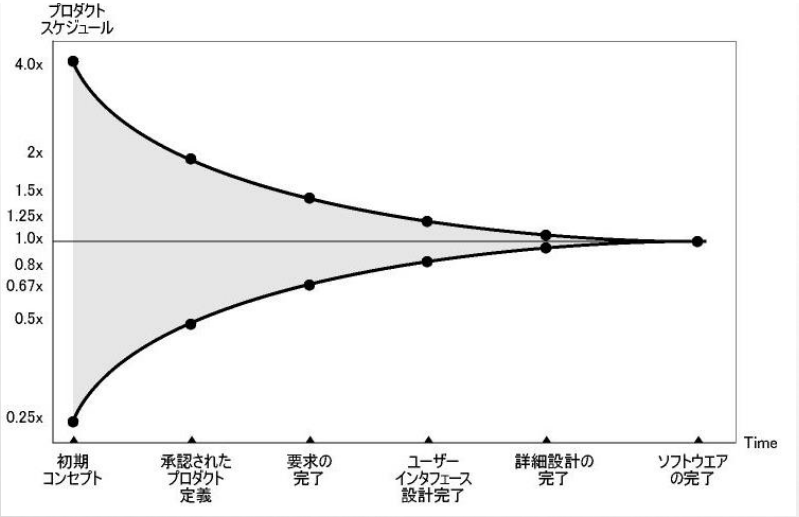
変更指示手順の複雑なプロセスが、顧客を疲弊させやがて信頼を失わせる
変更管理プロセスが優れていても、そのプロセスは遅かれ、早かれ顧客の信頼を失うことになる

＜理由１＞
顧客からすれば、請負業者は最初から要求をすべて理解して当然と考えている
要求を再び説明するのに時間をかけ、長い変更管理プロセスを経る
コストはかさみ、納期は遅れる
そうやってプロセスを通過しても、製品は相変わらず思ったとおりに出てこないかもしれないがコストは払わなければならない

＜理由２＞
多くの会社は顧客が心変わりすると知っており、入札の際は最小の金額で入札する
伝統的な契約では請負業者がすべてリスクを取ることになっていたため、変更プロセスが必須
顧客は本当の要求をあとから見つけるという事実につけこみ、変更管理プロセスを儲けの種にするビジネス戦略になっている

タイミング

プロジェクトの最初から、顧客は納期、コストの確約を求める
このような要求をプロジェクトの最初というタイミングで出すのは契約の初めから顧客、請負業者の双方を悪い状況に追い込む



プロジェクトが始まったときは、不確実性は最大
不確実な状況でチームは正確な機能要件、納期、コストを
ブレなく見積もるように強制される

引用：プロジェクトマネジャーのための「プロセス設計術」 - プロジェクトの本質とはなにか：ITpro

スクラムでも最終的なコスト、機能、納期を正確に予測することはできない
変更を受け入れ、プロジェクトの初期の不確実な範囲での確約を制限し、早い時期での学習を最大化するこれまでとは違う契約モデルが必要

範囲と変更モデル

＜伝統的なソフトウェアプロジェクト契約＞
　　マイルストン、成果記述、プロジェクトのゴールや目的、導入スケジュール、保証
＜範囲と変更契約＞
　　上記と同じ要素を含むこともあるがやり方が大きく異なる
　　発見契約とプロジェクト契約の2つの契約でカバーされる
　　最初の契約は発見フェーズしかカバーしない
■発見契約：範囲の確認
　　発見フェーズは、固定コスト、固定期間の契約でただひとつのゴールを持つ
　　チームが必要な情報を集め、以下を発見し、顧客に伝える
　　　　何を届けられるか(プロダクトバックログ)
　　　　いつ届けられるか(初期スプリントリリース計画)
　　　　どれだけお金がかかるか(スプリントあたりのコスト)
　　2～3人の発見チームを組み、2週間で実施することが多い
　　発見契約の終了後、顧客が他の請負業者を使うことにしても成果物は利用できる
　　　　＜プロダクトバックログ＞
　　　　　　ステップ1．ユーザーのタイプもしくはペルソナを特定する
　　　　　　ステップ2．ユーザーストーリーを書く
　　　　　　ステップ3．ストーリーを見積もる
■発見フェーズ：コストとタイムラインの確定
　　チームが顧客と一緒に働き、情報が集まったら、コストとタイムラインを決める
　　　　＜初期スプリントリリース計画/スプリントあたりのコスト＞
　　　　　　ステップ1．チームのベロシティを決める(スライド17)
　　　　　　ステップ2．スプリントのコストを計算する
　　　　　　ステップ3．リリース計画をつくる(スライド27)
　　　　　　ステップ4．支払いオプションを確定する
　　　　　　　　単価をポイントあたりで設定する(範囲を設定してもよい)
　　　　　　　　スプリントの最低ポイントを保証したうえでスプリントあたりの単価を設定する

範囲と変更モデル

■プロジェクト契約：納品

発見フェーズが完了すると成果物が提供できる状態になる

＜発見フェーズでの成果物＞

ユーザーペルソナのセット
見積もられたプロダクトバックログ
リリース計画
プロジェクト契約

＜プロジェクト契約＞

請負業者があるポイントの範囲の機能をスプリント毎に一定コストで届けると保証する
チームの完成の定義や顧客への納品を方法を記述する
顧客がスプリントの成果を拒否できる条件について記述する
変更への対応するための変更条項を記述する

＜変更＞

スクラム計画でも変更条項はある
2008年アジャイルカンファレンスでジェフ・サザーランドのコンセプト説明
『①納品がなくても有料。②変更は無料』

「変更は無料」
顧客にチームと一緒に働くことを求め、プロダクトオーナーが優先順位づけに責任を負うように求めている
同じくらいのサイズのストーリーを取り除くことで、無料で新しいストーリーを足すことができる

範囲と変更モデル

契約が良い情報と顧客との協働に基づいていれば、顧客との信頼関係を築くまでの長い道のりを歩み始められる
コストの幅や変更の許容範囲以外にも、スクラムプロジェクトを進めたり、壊したりする要素について契約に含むべきである

＜顧客の参画(フィードバック、受け入り、プロダクトバックログのメンテナンス)＞
- 1週間あたり、顧客が何時間プロジェクトに参画するかを契約に含める
- スプリントが短い場合にはほぼ毎日の参画が必要になる
- スプリントが長いと顧客の参画がプロジェクトの始めと終わりに集中するというリスクがある

＜受け入れウィンドウ(受け入れ判断に使える許容時間)＞
- 顧客はこの時間内にスプリントで開発された機能を受け入れるか判断(理想的には一日以下)
- 受け入れウィンドウが経過するまでに受け入れも拒否しなかった場合のデフォルトの判断も決めておく
- 受け入れウィンドウ内で不足が見つかれば受け入れ拒否となり、請負業者が自分のコストで直す
- 顧客が受け入れを拒否すると、その仕事分の費用は支払われない。
- 受け入れ拒否を悪用するような信頼できない顧客は選ぶべきでなない
- 信頼できない顧客にノーという方法を学ぶ必要もある

＜優先順位づけ＞
- 顧客がプロダクトバックログを常にメンテナンスするための時間を契約に含める
- 開発の流れの都合で、チームが順序を決めたい場合、順序によって見積もりが変わることがあることを顧客に理解してもらう

＜契約終了事項＞
- 顧客はいつでもいかなる理由でも契約を終了できると書いておいたほうがいい
- 実行中のスプリントが終わり、もう1スプリントでコードベースの移動、書類の処理など必要な片付け作業が終われればプロジェクトを終了する
- これは信頼を醸成するし、どのスプリントでもコードを出荷可能にしておき必要があるとチームが理解できる

「納品がなくても有料」
- リストの残りを実装するのにメリットがないとわかった場合、顧客は残りの仕事を放棄する
- 残りのプロジェクトで想定していた金額の一部を請負業者が、契約早期終了のコストをカバーするために受け取る

＜信頼＞
- 契約で変更を受け入れられる構造にするのが、ビジネスではベスト
- 顧客が都度払いするオプションを準備、同じサイズのストーリーを出し入れする自由を認め、早期終了の場合、双方にメリットがあるようにする
- ソフトウェア開発企業にとって、信頼よりも重要なものはない